G&C 30566.315-US-01

*Patent Application*

*for:*

# STACK-BASED CALLBACKS FOR DIAGNOSTIC DATA GENERATION

*Inventors:*

**Ravinder P. Krishnaswamy, Ashok K. Gadangi and Davis C. Augustine**

*Prepared by:*

Gates & Cooper LLP

Howard Hughes Center

6701 Center Drive West, Suite 1050

Los Angeles, California 90045

# STACK-BASED CALLBACKS FOR DIAGNOSTIC DATA GENERATION

## BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention generally relates to a system for diagnosing program failure, and, in particular, to a stack-based callback scheme for diagnostic data generation.

2. Description of the Related Art

Software programs often fail by "crashing" or reaching error conditions that cause them to terminate. In order to improve product quality, it is important to diagnose the reasons for failure.

Operating systems often generate crash data for software programs, wherein the crash data can be analyzed in an attempt to diagnose the reasons for failure. For example, MICROSOFT WINDOWS operating systems create a "full dump" or "minidump" file, and UNIX or LINUX operating systems create a "core dump" file, when a program terminates due to unhandled error conditions.

It is well known for software program vendors to provide users with a set of tools for capturing and analyzing program crash data. In their simplest form, these tools comprise an error reporting mechanism that presents the users with an alert message that notifies them when a failure occurs and provides an opportunity to forward crash data to the vendor for further analysis. The vendor can then use the forwarded crash data to troubleshoot problems, ultimately leading to more robust and crash-resistant programs.

Often, the programs are sufficiently complex, so that even though the point of failure is known, the amount of information that captures the program's full state in the crash data is unmanageably large, e.g., 100's of Megabytes. Even then, the crash data may not contain all the information needed to diagnose the problem. For example, some of the information needed to diagnose the problem may be external to the program, e.g., stored in files. Also, the information needed to diagnose the problem is often only a small fraction of the crash data, e.g., 10's of Kilobytes. Although there are sometimes options to generate subsets of the program's full state in the crash data, such subsets are almost certain to miss critical diagnostic information.

The key point is that the crash data generated by an operating system is usually directed at a generic program, since the operating system does not know about specific internal logic and critical state data pertaining to the program.

Thus, there is a need in the art for a mechanism where the information to help diagnose the problem can be intelligently supplied by the program itself. Specifically, there is a need in the art for a mechanism that uses the program's knowledge of what information is relevant at the point of failure when providing the crash data to help diagnose the program. The present invention satisfies that need.

## SUMMARY OF THE INVENTION

To address the requirements described above, the present invention discloses a method, apparatus, and article of manufacture for implementing a stack-based callback scheme in a software program to acquire diagnostic information. The five phases of the scheme include Callback Registration, Stack Determination, Stack-based Callback Notification, Callback Processing and Diagnostic Data Packaging.

## BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 schematically illustrates an exemplary hardware and software environment used in the preferred embodiment of the present invention; and

FIG. 2 is a flowchart that further illustrates the functions performed by a program according to the preferred embodiment of the present invention.

## DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the following description, reference is made to the accompanying drawings which form a part hereof, and which is shown, by way of illustration, several embodiments of the present invention. It is understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the present invention.

Overview

The present invention describes a mechanism to provide contextual diagnostic data at the point of failure of a software program by explaining the implementation logic of a stack-based callback scheme for diagnostic data generation. The goal of the present invention is a mechanism where the data to help diagnose the problem can be intelligently supplied by the program itself by building logic into the program to make use of the call stack context as much as possible in determining relevant diagnostic data. This means it also provides the ability for programs to include data external to the program, such as files.

Hardware and Software Environment

FIG. 1 schematically illustrates an exemplary hardware and software environment used in the preferred embodiment of the present invention. The present invention is usually implemented using a network 100 to connect one or more workstations 102 to one or more server computers 104. A typical combination of resources may include workstations 102 that comprise personal computers, network computers, etc., and server computers 104 that comprise personal computers, network computers, workstations, minicomputers, mainframes, etc. The network 100 coupling these computers 102 and 104 may comprise a LAN, WAN, Internet, etc.

Generally, the present invention is implemented using one or more programs, files and/or databases that are executed and/or interpreted by the workstations 102. In the exemplary embodiment of FIG. 1, these programs and databases include one or more workstation programs 106 executed by one or more of the workstations 102, and context data 108 stored on a data storage device 110 accessible from the workstation 102. In addition, the environment often includes one or more server programs 112 executed by the server computer 104, and a crash database 114 stored on a data storage device 116 accessible from the server computer 104.

Each of the programs and/or databases comprise instructions and data which, when read, interpreted, and executed by their respective computers, cause the computers to perform the steps necessary to execute the steps or elements of the present invention. The programs and databases are usually embodied in or readable from a computer-readable device, medium, or carrier, e.g., a local or remote data storage device or memory device

coupled to the computer directly or coupled to the computer via a data communications device.

Thus, the present invention may be implemented as a method, apparatus, or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively, "computer program carrier or product") as used herein is intended to encompass one or more programs and/or databases accessible from any device, carrier, or media.

Of course, those skilled in the art will recognize that the exemplary environment illustrated in FIG. 1 is not intended to limit the present invention. Indeed, those skilled in the art will recognize that other alternative environments may be used without departing from the scope of the present invention.

Stack-based Callback

FIG. 2 is a flowchart that further illustrates the functions performed according to the preferred embodiment of the present invention. Specifically, the flowchart shows the five phases performed by the present invention to implement a stack-based callback in a program to acquire diagnostic information.

Block 200 represents the step of Registration. The program typically is comprised of different modules and sub-applications. This Block comprises registering callbacks for one or more of the different modules and sub-applications within the program when an address of a procedure or function within the modules and sub-applications is on a call stack upon the failure of the program. Generally, the Registration step is performed when the program starts, and when new modules and sub-applications are loaded by the program while the program is running.

Block 202 represents the step of Stack Determination. This Block comprises an error handler for the program that determines the call stack for the program at the point of failure of the program.

Block 204 represents the step of Callback Notification. This Block comprises the error handler for the program notifying the registered callbacks of the modules and sub-applications based on the examined call stack. In this Block, the error handler gives each

module or sub-application that registered a callback function and is on the failure call stack an opportunity to include specific diagnostic information based on the call stack and based on the point of failure within the module or sub-application.

Block 206 represents the step of Callback Processing. This Block comprises the processing logic within the callbacks for the modules and sub-applications registered in Block 200 and invoked in Block 204. Each individual module and sub-application callback processing logic involves interpreting the failure call stack context supplied to the callback function and determining relevant contextual data to be supplied to the error handler of the program 106 for packaging in Block 208. The context data is generally extracted from stack data, heap data, global data and external data. The stack data is usually comprised of local variables as well as addresses of (i.e., references to) functions, variables and objects, while the external data may comprise files or any other kind of relevant data that a specific module or sub-application wants to attach as part of the context.

Block 208 represents the step of Diagnostic Data Packaging. This Block comprises the error handler for the program 106 packaging the context data supplied by the notified modules and sub-applications for examination and/or delivery. This Block may also include storing the packaged context data 108 on the data storage device 110 accessible from the workstation 102, transferring the packaged context data 108 from the program 106 to the server program 112, and/or storing the packaged context data 108 in the crash database 114 on the data storage device 116 accessible from the server computer 104.

Conclusion

This concludes the description of the preferred embodiment of the invention. The following describes some alternative embodiments for accomplishing the present invention.

For example, any type of computer, such as a mainframe, minicomputer, work station or personal computer, or network could be used with the present invention. In addition, any program, application or operating system could benefit from the present invention. It should also be noted that the specific programs described herein are not intended to limit the invention, but merely to provide examples.

The foregoing description of the preferred embodiment of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive

or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto.